



INTEL ADVISOR: ROOFLINE AUTOMATION

Intel Software and Services, 2017

Zakhar Matveev, PhD, Product architect

5 Steps to Efficient Vectorization - Vector Advisor

(part of Intel® Advisor, Parallel Studio, Cluster Studio 2016)

1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time	Total Time	Compiler Vectorization
[0] loop in runCforallLambdaLoops	0.094s	0.094s	Scalar vector dependence prevents vector...
[0] loop in runCforallLambdaLoops	0.140s	3.744s	Scalar inner loop was already vectorized
[0] loop in std::complex_base<double,struct _C_double_complex>::...	0.031s	0.031s	Vectorized (Both)

Vectorized SSE: SSE2 loop processing Float32; Float64 data type
Peeled loop: loop stats were reordered

Function Call Sites and Loops	Self Time	Total Time
[0] loop in std::basic_string<char,struct std::char_traits<char>,class std::allocator<char>>::...	0.000s	0.000s
[0] loop in std::basic_string<char,struct std::char_traits<char>,class std::allocator<char>>::...	0.000s	0.000s
[0] loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s	0.000s

2. Guidance: detect problem and recommend how to fix it

Issue: Peeled/Remainder loop(s) present
All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at [Vector Essentials: Utilizing Full Vectors...](#)

Recommendation: Align memory access
Projected maximum performance gain: High
Projection confidence: Medium

Use one of the memory accesses in the source loop does not try access and tell the compiler your memory access is aligned, byte boundary.

```
SIZE*sizeof(float), 32);
```

3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

Total Time	Trip Counts			Iteration Duration	Call Count
	Median	Min	Max		
3.151s	1	1	1	3.1509s	1
0.440s	1	1	1	< 0.0001s	2408000
0.010s	1	1	2	< 0.0001s	207596
0.010s	1	2	9	< 0.0001s	1173619
0.010s	1	3	5	< 0.0001s	1312315

4. Loop-Carried Dependency Analysis

Problems and Messages

ID	Type	Site Name	Sources	Modules	State
P1	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	Read after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P3	Read after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P4	Write after write dependency	site2	dqtest2.cpp	dqtest2	✗ New
P5	Write after read dependency	site2	dqtest2.cpp	dqtest2	✗ New
P6	Write after read dependency	site2	dqtest2.cpp	dqtest2	✗ New
P7	Write after read dependency	site2	dqtest2.cpp, idle.h	dqtest2	✗ New

5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runRawLoops	runRawLoops.coc1063	RAW:1	No information available	No information available
loop_site_139	runRawLoops	runRawLoops.coc622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runRawLoops	runRawLoops.coc925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns

ID	Stride	Type	Source	Modules	Alignment
P22	0; 0; 1	Unit stride	runRawLoops.coc637	lcal.exe	
P23	0; 0	Unit stride	runRawLoops.coc638	lcal.exe	
P30	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runRawLoops.coc628	lcal.exe	

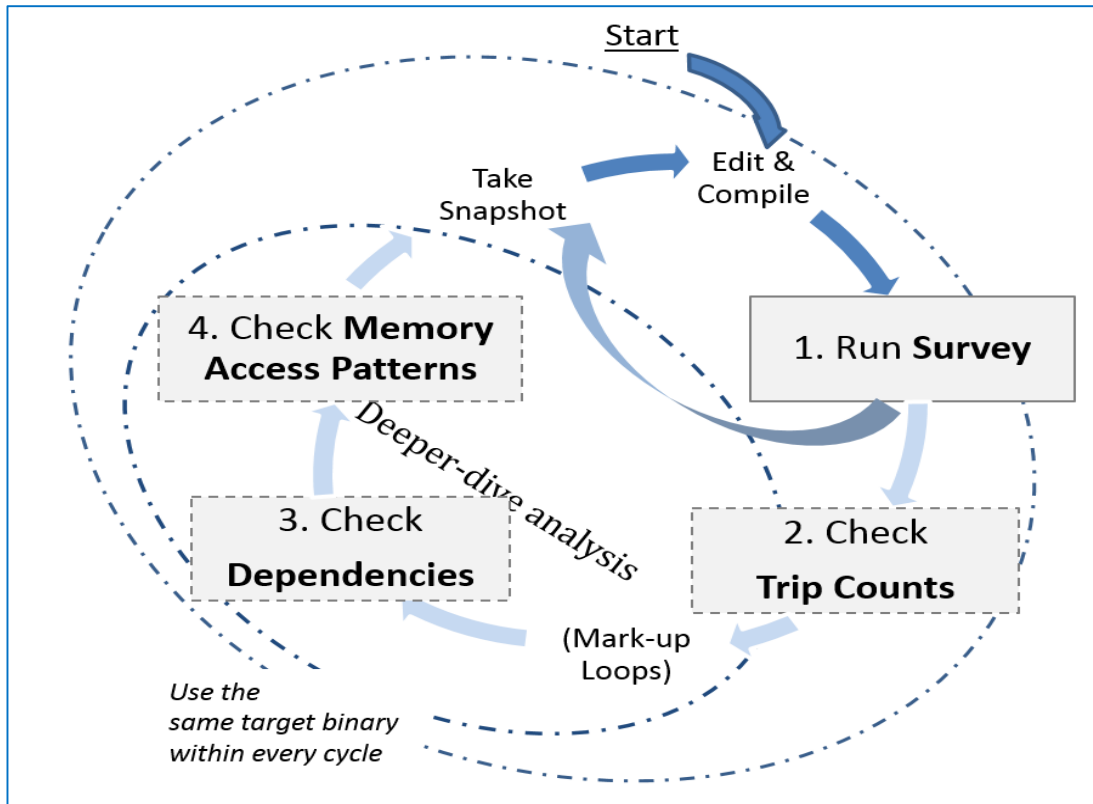
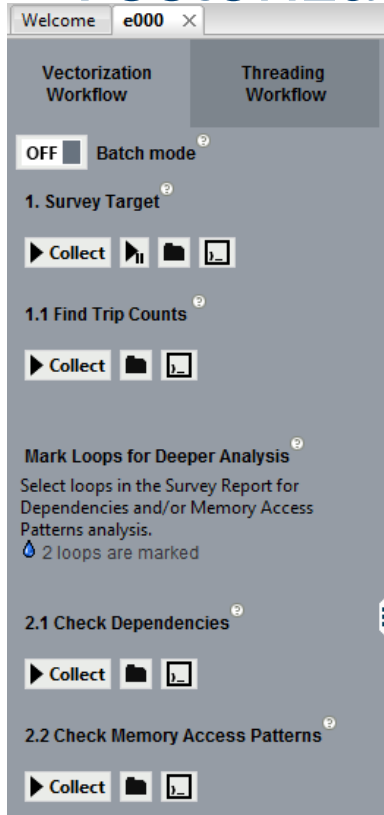
```
635 j2 = ( j2 < 64-1 ) ;  
636 p[ip][0] += y[j2+32];  
637 p[ip][1] += x[j2+32];  
638 i2 += e[j2+32];  
639 j2 += f[j2+32];
```

```
626 i1 <= 64-1;  
627 j1 <= 64-1;  
628 p[ip][2] += b[j1][i1];
```

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Vectorization Analysis Workflow



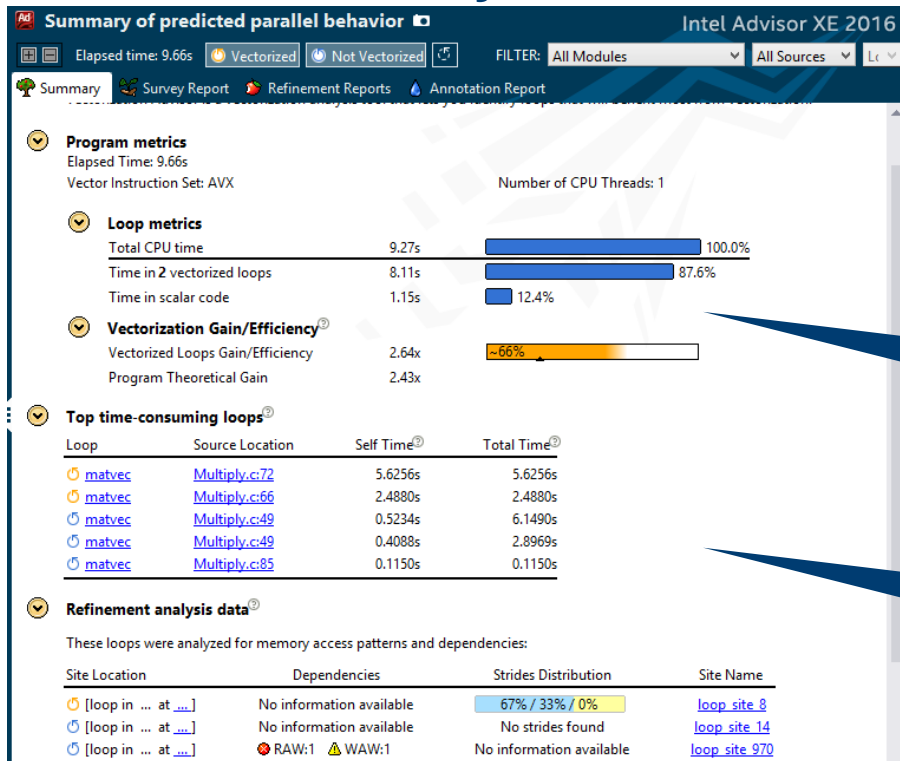
Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Quickly **characterize** the efficiency of your code: Advisor Summary.



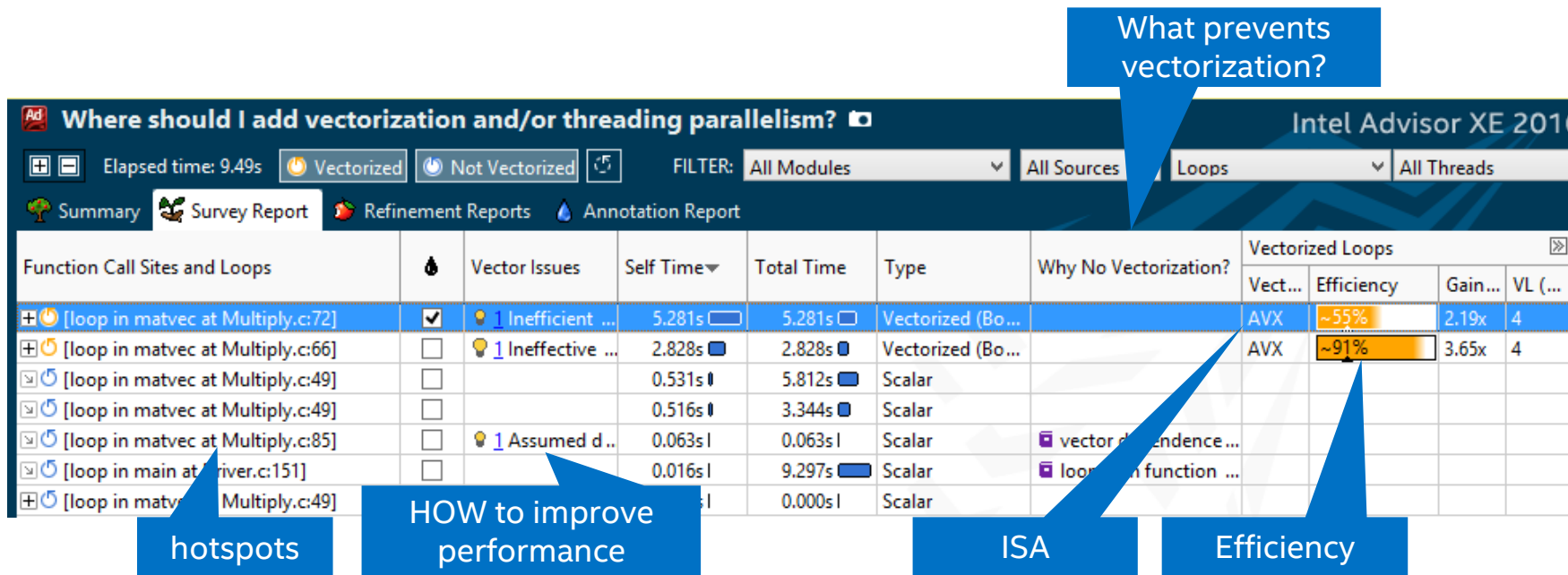
Use the summary view to quickly characterize your program

Time in
Scalar vs. Vector loops.
SIMD Efficiency.

Focus on Hottest
kernels

Advisor Survey: Focus + Characterize.

One stop shop.



All the data you need for effective vectorization

Advisor Memory Access Pattern (MAP): know your access pattern

Unit-Stride access

```
for (i=0; i<N; i++)
  A[i] = C[i]*D[i]
```

Constant stride access

```
for (i=0; i<N; i++)
  point[i].x = x[i]
```

Variable stride access

```
for (i=0; i<N; i++)
  A[B[i]] = C[i]*D[i]
```

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in fPropagationSwap at lbpSUB.cpp:1247]	No information available	33% / 5% / 62%	Mixed strides	loop_site_60

blue color: fraction of unit stride accesses
 yellow: "fixed" stride accesses ratio
 red color: fraction of irregular (variable stride) accesses

Memory Access Patterns Report	Dependencies Report
-------------------------------	---------------------

ID	Stride	Type	Source	Site Name	Variable
P1	3	16% / 84% / 0%	Mixed strides		

```
1246 #endif
1247 for (int m=1; m<=half; m++) {
1248     nextx = fCppMod(i + lbv[3*m])
1249     nexty = fCppMod(j + lbv[3*m+1])
1250     nextz = fCppMod(k + lbv[3*m+2])
1251 }
```

```
P11 0; 1
```

```
P12 -289559; -274359; -14477; -13717; -13679; 723; 302519;
```

```
1251 ilnext = (nextx * Ymax + nexty * Xmax + nextz * Zmax) % MOD;
1252 #ifndef SWAP_OVERLAP
1253 fSwapPair (lbf[i]*lbsitelength + 1*lbsy...
```

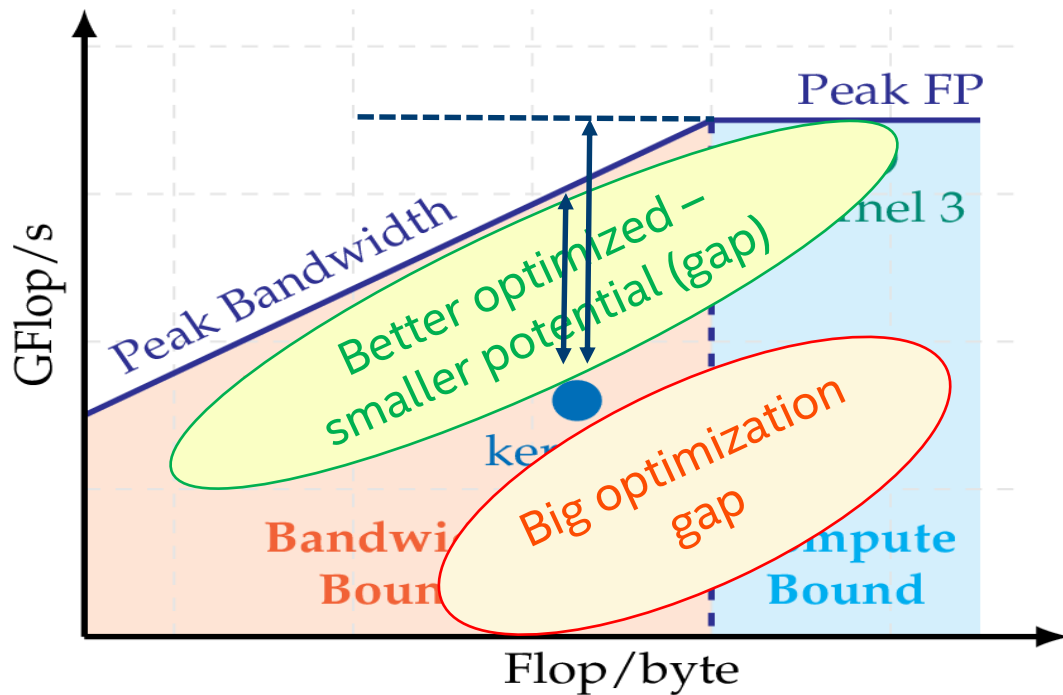
16%: percentage of memory instructions with unit stride or stride 0 accesses
 Unit stride (stride 1) = Instruction accesses memory that consistently changes by one element from iteration to iteration
 Stride 0 = Instruction accesses the same memory from iteration to iteration

84%: percentage of memory instructions with fixed or constant non-unit stride accesses
 Constant stride (stride N) = Instruction accesses memory by N elements from iteration to iteration
 Example: for the double floating point type, stride 4 means the memory address accessed by this instruction increased by 32 bytes, (4*sizeof(double)) with each iteration

0%: percentage of memory instructions with irregular (variable or random) stride accesses
 Irregular stride = Instruction accesses memory addresses that change by an unpredictable number of elements from iteration to iteration
 Typically observed for indirect indexed array accesses, for example, a[index[i]]
 - gather (irregular) accesses, detected for v(p)gather* instructions on AVX2 Instruction Set Architecture

Am I bound by VPU/CPU or by Memory?

ROOFLINE ANALYSIS



From “Old HPC principle” to modern performance model

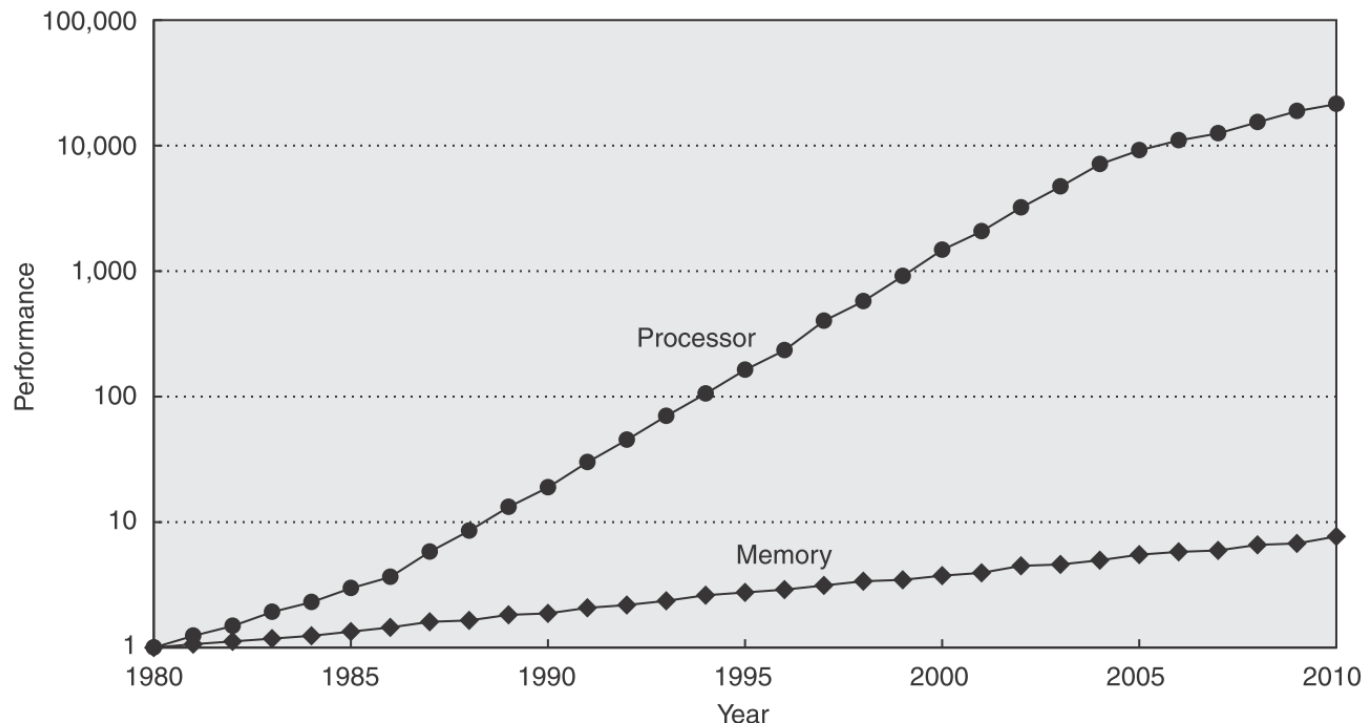
“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether
2. “Compute Density”, “intensity”, “machine balance” - (FLOP/byte or Byte/FLOP ratio for algorithm or hardware). E.g. Kennedy, Carr: 1988, 1994: “Improving the Ratio of Memory operations to Floating-Point Operations in Loops “.

More research catalyzed by memory wall/ gap growth and by GPGPU

- **2008, Berkeley:** generalized into Roofline Performance Model. Williams, Waterman, Patterson. “**Roofline: an insightful visual performance model for multicore**”
- **2014:** “Cache-aware Roofline model: ” Ilic, Pratas, Sousa. INESC-ID/IST, Technical Uni of Lisbon.

Memory Wall



Patterson, 2011

From “Old HPC principle” to modern performance model

“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether
2. “Compute Density”, “intensity”, “machine balance” - (FLOP/byte or Byte/FLOP ratio for algorithm). E.g. Kennedy, Carr: 1988, 1994: “Improving the Ratio of Memory operations to Floating-Point Operations in Loops “.

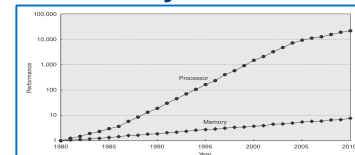
More research catalyzed by memory wall/ gap growth and by GPGPU:

- **2008, Berkeley:** generalized into Roofline Performance Model. Williams, Waterman, Patterson. **“Roofline: an insightful visual performance model for multicore”**
- **2014:** “Cache-aware Roofline model: ” Ilic, Pratas, Sousa. INESC-ID/IST, Technical Uni of Lisbon.

From “Old HPC principle” to modern performance model

“Old” HPC principles:

1. “Balance” principle (e.g. Kung 1986) – hw and software parameters altogether
2. “intensity”, “machine balance” - (FLOP/byte or Byte/FLOP ratio for algorithm or hardware). E.g. Kennedy, Carr: 1988, 1994: “Improving the Ratio of Memory operations to Floating-Point Operations in Loops “.



More research catalyzed by memory wall

- 2008, Berkeley: generalized into Roofline Performance Model. Williams, Waterman, Patterson. “Roofline: an insightful visual performance model for multicore”
- 2014: “Cache-aware Roofline model: ” Ilic, Pratas, Sousa. INESC-ID/IST, Technical Uni of Lisbon.

Density, Intensity, Machine balance

$$\text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred}}$$

WIP

AI

$$\text{Arithmetic Operational Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred between DRAM (MCDRAM) and LLC}}$$

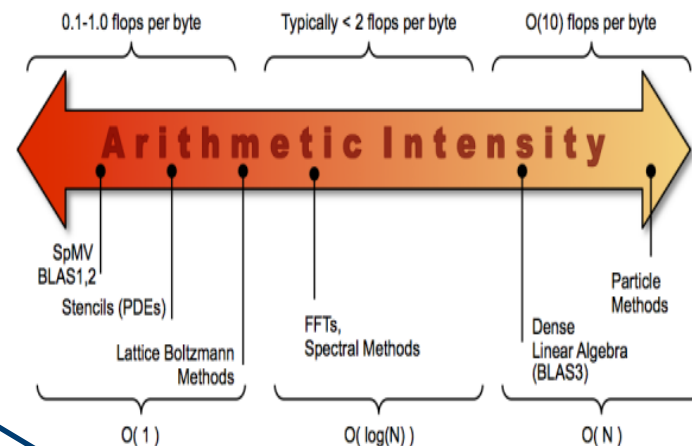
Implemented in 2017 Update 1

AI

$$\text{Arithmetic Intensity} = \frac{\text{Total Flops computed}}{\text{Total Bytes transferred between CPU and "memory"}}$$

WIP

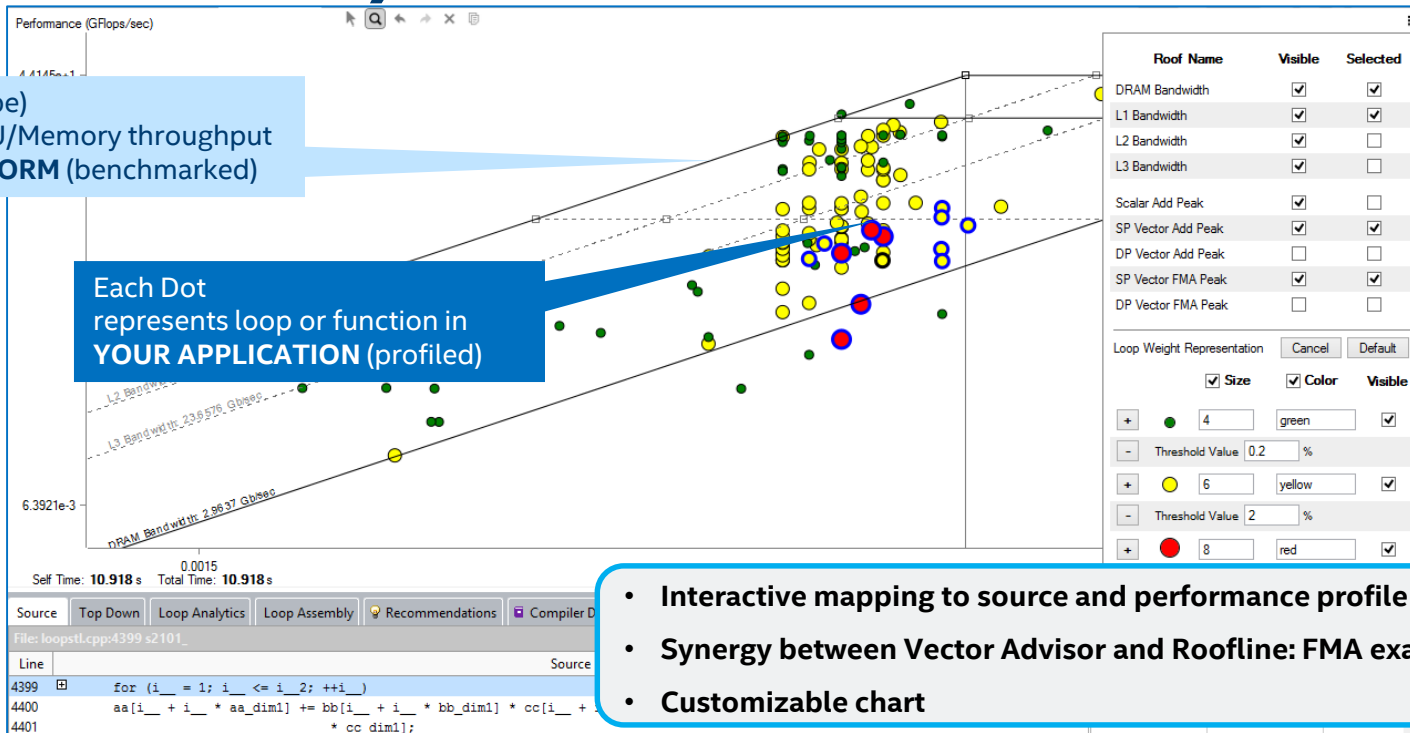
$$\text{Arithmetic Intensity} = \frac{\text{Total Intops+Flops computed}}{\text{Total Bytes transferred between CPU and "memory"}}$$



Roofline Automation in Intel® (Vectorization) Advisor 2017

Each Roof (slope)
Gives peak CPU/Memory throughput
of your **PLATFORM** (benchmarked)

Each Dot
represents loop or function in
YOUR APPLICATION (profiled)

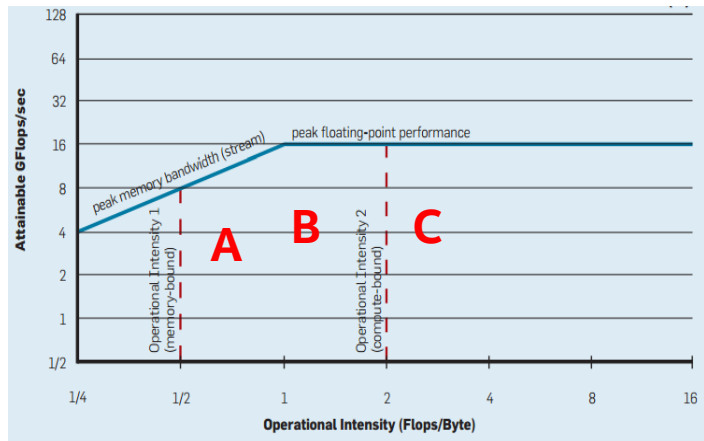


- Interactive mapping to source and performance profile
- Synergy between Vector Advisor and Roofline: FMA example
- Customizable chart

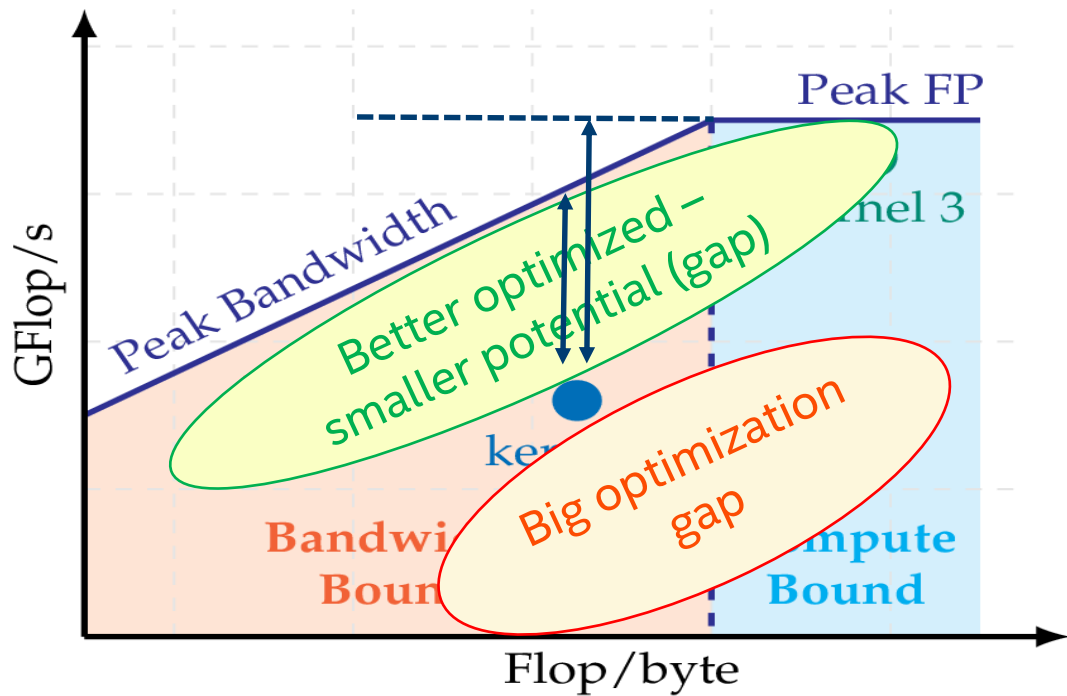
Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Roofline model: Am I bound by VPU/CPU or by Memory?



What makes loops
A, B, C different?



Advisor Roofline: under the hood

Roofline application profile:

Axis Y: $\text{FLOP/S} = \# \text{FLOP (mask aware)} / \# \text{Seconds}$

Axis X: $\text{AI} = \# \text{FLOP} / \# \text{Bytes}$

Seconds

User-mode **sampling**

Root access not needed

Performance = Flops/seconds

#FLOP

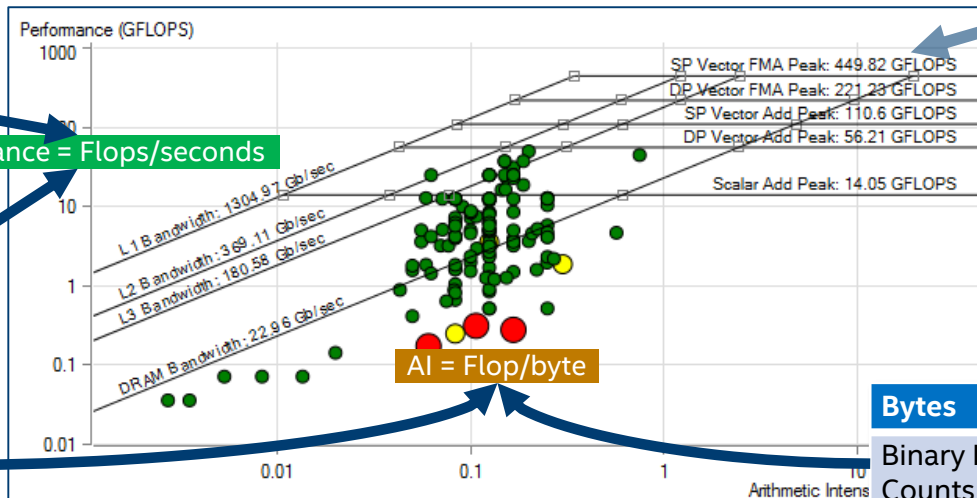
Binary **Instrumentation**

Does not rely on CPU counters

Roofs

Microbenchmarks

Actual peak for the current configuration



Bytes



Binary **Instrumentation**

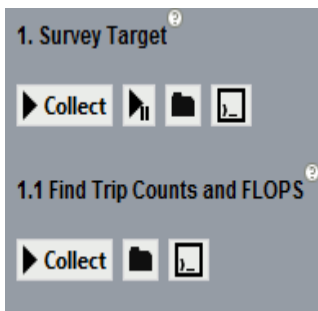
Counts operands size (not cachelines)

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Getting Roofline in Advisor

FLOP/S = #FLOP/Seconds	Seconds	#FLOP Count - Mask Utilization - #Bytes
Step 1: Survey <ul style="list-style-type: none">- Non intrusive. <i>Representative</i>- Output: Seconds (+much more)		
Step 2: FLOPS <ul style="list-style-type: none">- Precise, instrumentation based- Physically count Num-Instructions- Output: #FLOP, #Bytes		



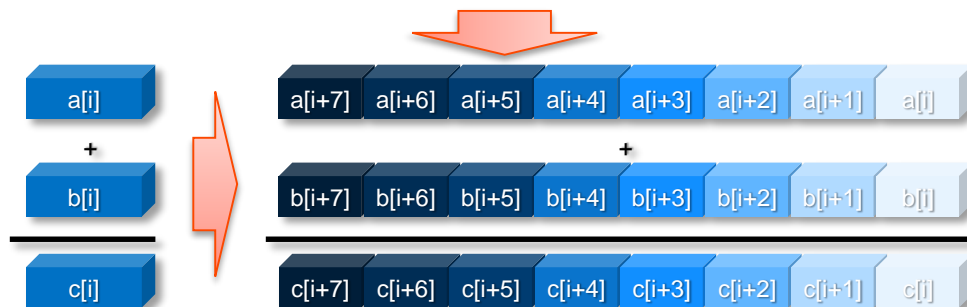
Mask Utilization and FLOPS profiler

- Long-waiting in HPC: accurate HW independent FLOPs measurement tool
- Not just count FLOPs. Has following additions:
 - (AVX-512 only) Mask-aware. Masked-Memory/Unmasked-Compute pattern aware
 - Unique capability to correlate FLOPs with performance data (obtained without instrumentation). Gives FLOPs/s.
- Lightweight instrumentation, PIN-based, benefits from “threadchecker tools” and more generally Advisor framework integration.

Why Mask Utilization Important?

```
for(i = 0; i <= MAX; i++)  
  c[i] = a[i] + b[i];
```

100%

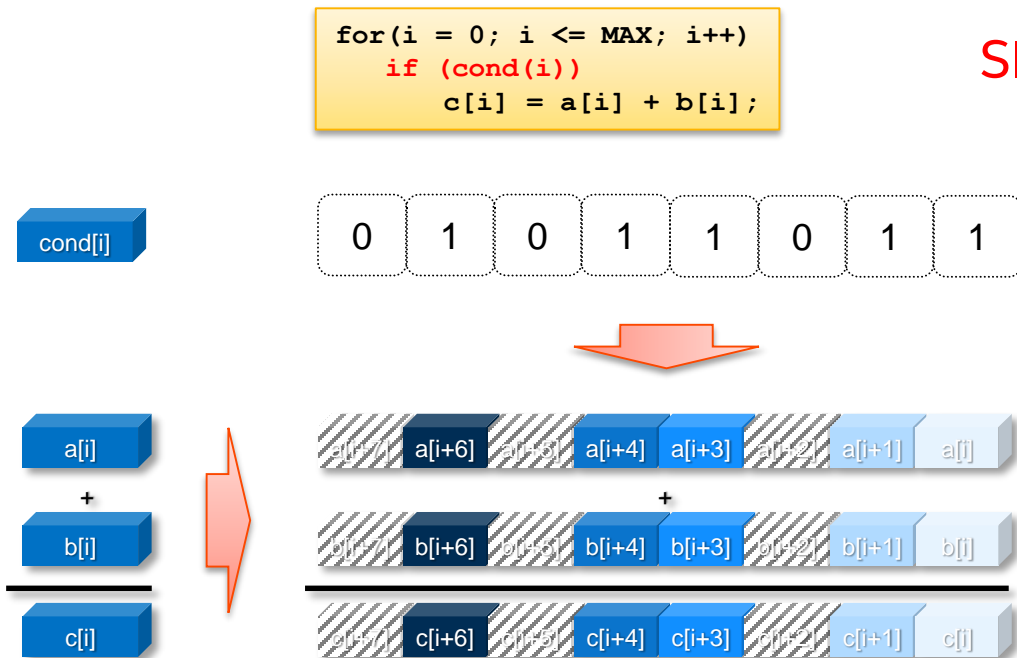


Why Mask Utilization Important?

3 elements suppressed

SIMD Utilization = 5/8

62.5%



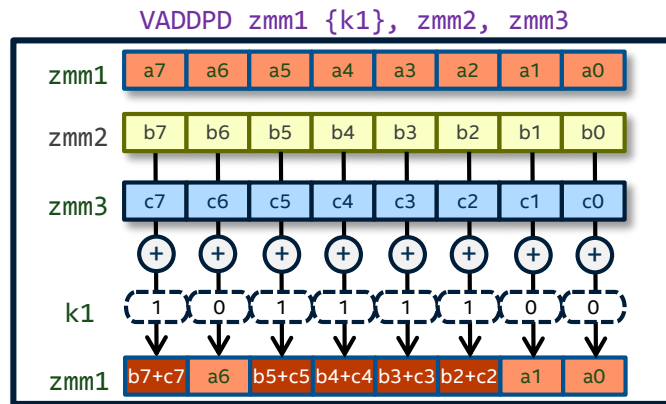
AVX-512 Mask Registers

8 Mask registers of size 64-bits

- k1-k7 can be used for predication
 - k0 can be used as a destination or source for mask manipulation operations

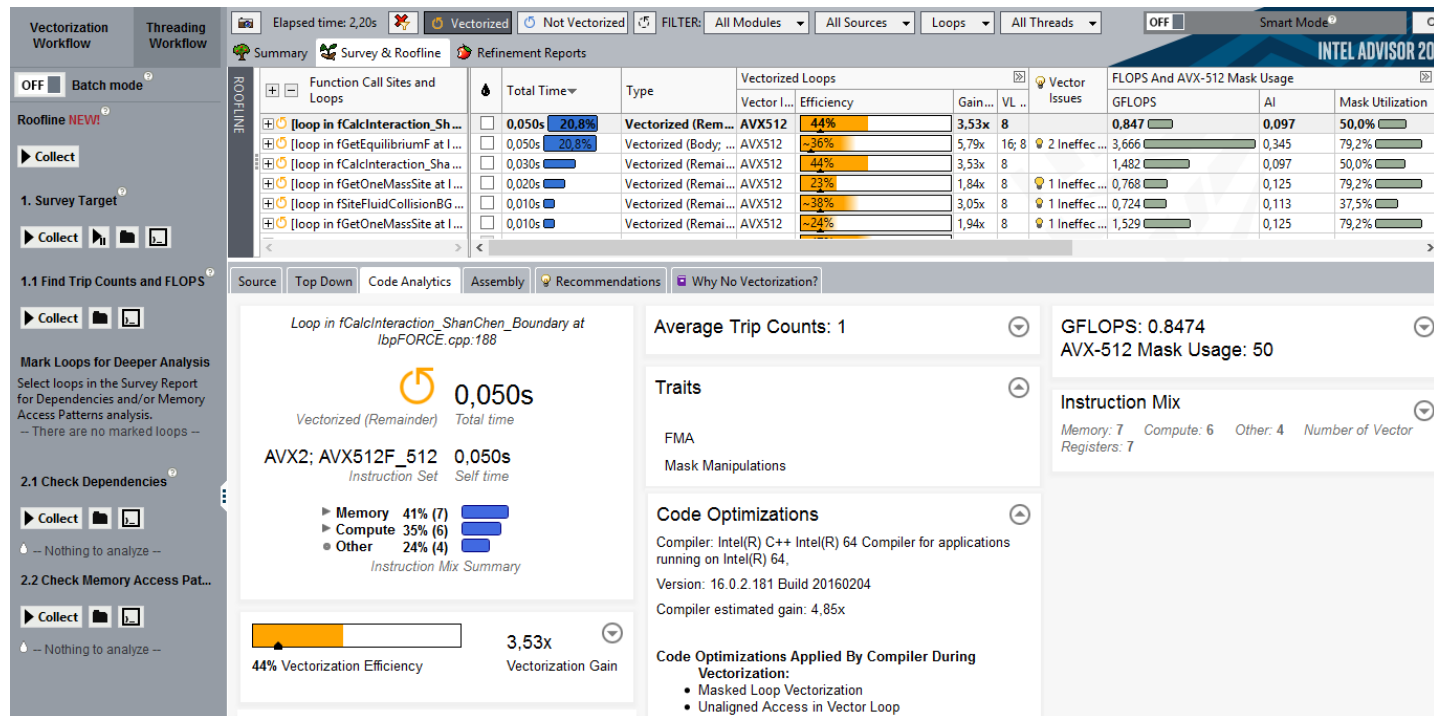
4 different mask granularities.
For instance, at 512b:

- Packed Integer Byte use mask bits [63:0]
 - `VPADDB zmm1 {k1}, zmm2, zmm3`
- Packed Integer Word use mask bits [31:0]
 - `VPADDW zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
 - `VADDPS zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
 - `VADDPD zmm1 {k1}, zmm2, zmm3`



element size		Vector Length		
		128	256	512
	Byte	16	32	64
	Word	8	16	32
	Dword/SP	4	8	16
	Qword/DP	2	4	8

Survey+FLOPs Report on AVX-512: FLOP/s, Bytes and AI, Masks and Efficiency



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



General efficiency (FLOPS) vs. VPU-centric efficiency (Vector Efficiency)

Function Call Sites and Loops		Total Time▼	Type	Vectorized Loops				Vector Issues	FLOPS And AVX-512 Mask Usage		
				Vector I...	Efficiency	Gain...	VL ..		GFLOPS	AI	Mask Utilization
+	🔥 [loop in fCalcInteraction_Sh...	0,050s 20,8%	Vectorized (Rem...	AVX512	44%	3,53x	8	0	0,847	0,097	50,0%
+	🔥 [loop in fGetEquilibriumF at I...	0,050s 20,8%	Vectorized (Body; ...	AVX512	~36%	5,79x	16; 8	2 Ineffec...	3,666	0,345	79,2%
			ectorized (Remai...	AVX512	44%	3,53x	8	0	1,482	0,097	50,0%
			ectorized (Remai...	AVX512	23%	1,84x	8	1 Ineffec...	0,768	0,125	79,2%
			ectorized (Remai...	AVX512	~38%	3,05x	8	1 Ineffec...	0,724	0,113	37,5%
			ectorized (Remai...	AVX512	~24%	1,94x	8	1 Ineffec...	1,529	0,125	79,2%

High Vector Efficiency
Low FLOPS

Function Call Sites and Loops		Total Time	Type	Vectorized Loops				Vector Issues	FLOPS And AVX-512 Mask Usage		
				Vector I...	Efficiency	Gain...	VL ..		GFLOPS	AI	Mask Utilization
+	[loop in fCalcInteraction_Sha ...	0,050s 20,8%	Vectorized (Remai...	AVX512	44%	3,53x	8		0,847	0,097	50,0%
+	[loop in fGetEquilibriumF at I ...	0,050s 20,8%	Vectorized (Body; ..	AVX512	~36%	5,79x	16; 8	2 Ineffec ...	3,666	0,345	79,2%
			Vectorized (Remai...	AVX512	44%	3,53x	8		1,482	0,097	50,0%
			Vectorized (Remai...	AVX512	23%	1,84x	8	1 Ineffec ...	0,768	0,125	79,2%
			Vectorized (Remai...	AVX512	~38%	3,05x	8	1 Ineffec ...	0,724	0,113	37,5%
			Vectorized (Rem...	AVX512	~24%	1,94x	8	1 Ineffe...	1,529	0,125	79,2%

Low Vector Efficiency

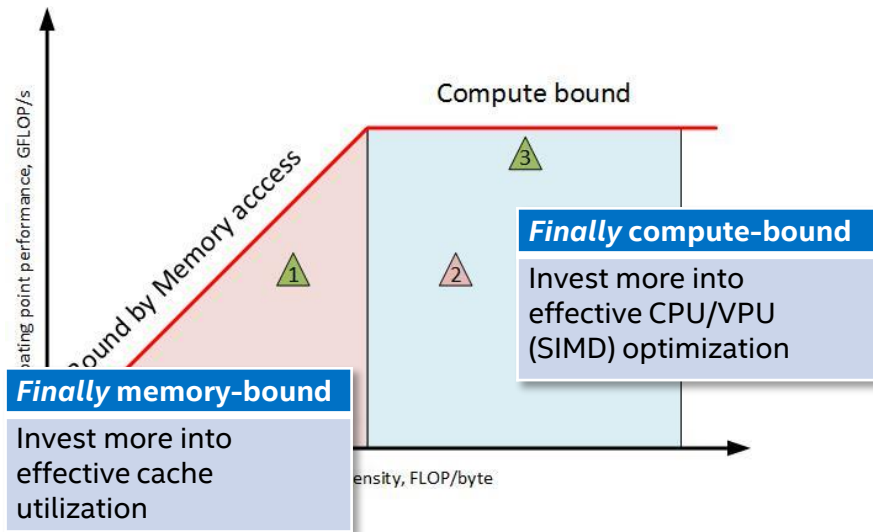
High FLOPS

Interpreting Roofline Data: advanced ROI analysis.

Final Limits

(assuming perfect optimization)

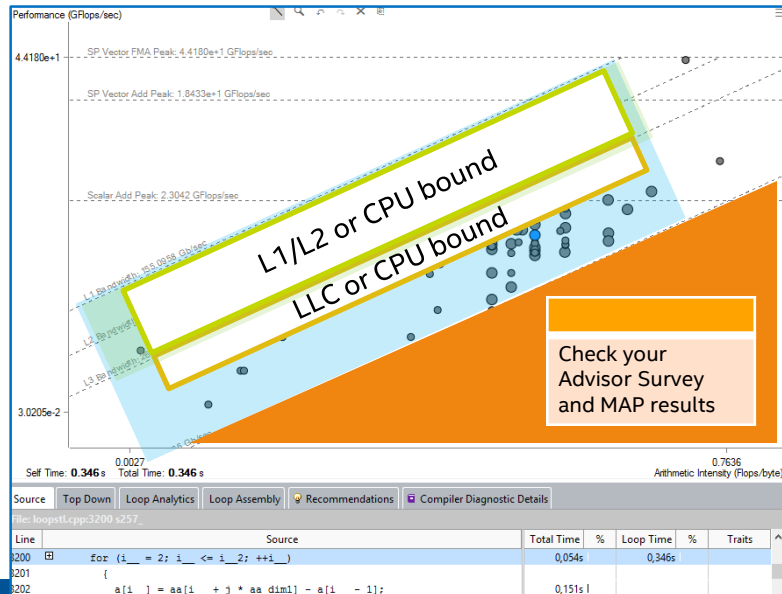
Long-term ROI, optimization strategy



Current Limits

(what are my current bottlenecks)

Next step, optimization tactics



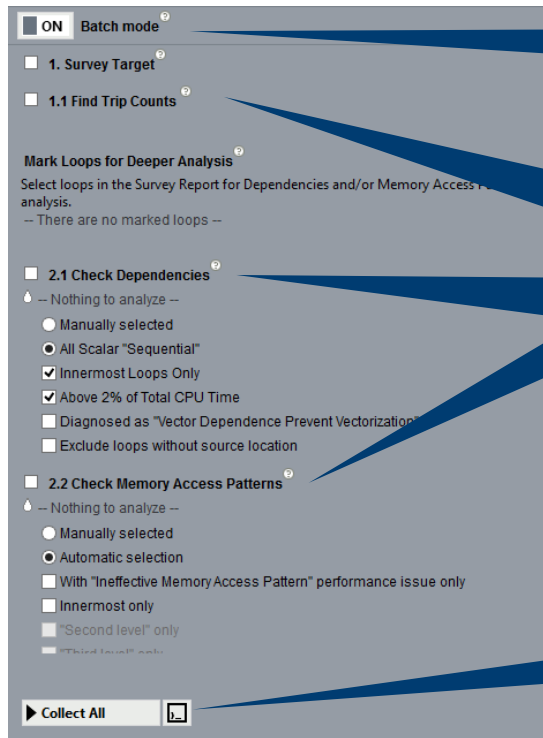
Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

BACKUP

Batch Mode Workflow Saves Time

Intel® Advisor - Vectorization Advisor



Turn On
Batch Mode

Run several analyses in batch
as a single run

Select
analyses to
run

Contains pre-selected criteria
for advanced analyses

Click
Collect all

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

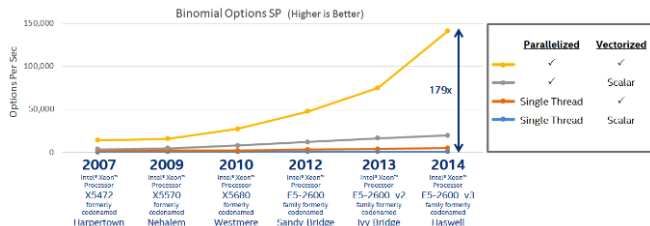
Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Configurations for Binomial Options SP



Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Performance measured in Intel Labs by Intel employees

Platform Hardware and Software Configuration

Platform	Unscaled Core Frequency	Cores/Socket	Num Sockets	L1 Data Cache	L1 I Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S Name	Operating System	Compiler Version
Intel® Xeon™ 5472 Processor	3.0 GHZ	4	2	32K	32K	12 MB	None	32 GB	800 MHZ	UMA	Y	N	N	Disable d	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5570 Processor	2.93 GHZ	4	2	32K	32K	256K	8 MB	48 GB	1333 MHZ	NUMA	Y	Y	Y	Disable d	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	32K	256K	12 MB	48 MB	1333 MHZ	NUMA	Y	Y	Y	Disable d	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	32K	256K	20 MB	64 GB	1600 MHZ	NUMA	Y	Y	Y	Disable d	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	32K	256K	30 MB	64 GB	1867 MHZ	NUMA	Y	Y	Y	Disable d	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Codename Haswell	2.2 GHz	14	2	32K	32K	256K	35 MB	64 GB	2133 MHZ	NUMA	Y	Y	Y	Disable d	Fedora 20	3.13.5-202.fc20	icc version 14.0.1

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



